Ordering Information:

Python How to Program
The Complete Python Training Course

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ONLINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html**.

To learn more about our **Python programming courses** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training**, contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com.**

Note from the Authors: This article is an excerpt from Chapter 16, Section 16.5 of *Python How to Program*. In this article, we discuss how to process an XML document using the Simple API for XML (SAX). We present a small XML document and implement a Python class that can parse the document with the SAX, using features from the Python standard library. The Python program enables the user to enter the filename of an XML document and a tag name and then searches the document for any matching tags. Readers should be familiar with basic python, exception handling, object-oriented programming (e.g., classes and inheritance) and XML. The code examples included in this article show readers programming examples using the DEITEL™ signature LIVE-CODE™ Approach, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

16.5 Parsing XML with `xml.sax`

In this section, we discuss the **`xml.sax`** package,[1] which provides a set of modules for SAX-based parsing. With SAX-based parsing, the parser reads the input to identify the XML markup. As the parser encounters markup, the parser calls *event handlers* (i.e., methods). For example, when the parser encounters a start tag, the ***`startElement`*** event handler is called; when the parser encounters character data, the ***`characters`*** event handler is called. Programmers override event handlers to provide specialized processing of the XML. Some common SAX event handlers are shown in Fig. 16.15.

Event Handler	Description
`characters(` *content* **`)`**	Called when the parser encounters character data. The character data is passed as *content* to the event handler.
`endDocument()`	Called when the parser encounters the end of the document.
`endElement(` *name* **`)`**	Called when the parser encounters an end tag. The tag *name* is passed as an argument to the event handler.
`startDocument()`	Called when the parser encounters the beginning of the document.
`startElement(` *name*, *attrs* **`)`**	Called when the parser encounters a start tag. The tag *name* and its attributes (*attrs*) are passed as arguments to the event handler.

Fig. 16.15 **`xml.sax`** event-handler methods.

Good Programming Practice 16.1

Review the Python on-line documentation for a complete listing of ***`xml.sax`*** *event handlers. This information can be found at:*

`www.python.org/doc/current/lib/content-handler-objects.html`

Figure 16.16 demonstrates SAX-based parsing. This program allows the user to specify a tag name to search for in an XML document. When the tag name is encountered, the program outputs the element's attribute-value pairs. Methods **`startElement`** and **`endElement`** are overriden to handle the events generated when start tags and end tags are encountered. Figure 16.17 contains the XML document used by this program.

```
1  # Fig. 16.16: fig16_16.py
2  # Demonstrating SAX-based parsing.
3
4  from xml.sax import parse, SAXParseException, ContentHandler
5
6  class TagInfoHandler( ContentHandler ):
7     """Custom xml.sax.ContentHandler"""
```

Fig. 16.16 SAX-based parsing example. (Part 1 of 3.)

1. Included in the standard library for Python versions 2.0 and higher.

```
 8
 9     def __init__( self, tagName ):
10        """Initialize ContentHandler and set tag to search for"""
11
12        ContentHandler.__init__( self )
13        self.tagName = tagName
14        self.depth = 0 # spaces to indent to show structure
15
16     # override startElement handler
17     def startElement( self, name, attributes ):
18        """An Element has started"""
19
20        # check if this is tag name for which we are searching
21        if name == self.tagName:
22           print "\n%s<%s> started" % ( " " * self.depth, name )
23
24           self.depth += 3
25
26           print "%sAttributes:" % ( " " * self.depth )
27
28           # check if element has attributes
29           for attribute in attributes.getNames():
30              print "%s%s = %s" % ( " " * self.depth, attribute,
31                 attributes.getValue( attribute ) )
32
33     # override endElement handler
34     def endElement( self, name ):
35        """An Element has ended"""
36
37        if name == self.tagName:
38           self.depth -= 3
39           print "%s</%s> ended\n" % ( " " * self.depth, name )
40
41  def main():
42     file = raw_input( "Enter a file to parse: " )
43     tagName = raw_input( "Enter tag to search for: " )
44
45     try:
46        parse( file, TagInfoHandler( tagName ) )
47
48     # handle exception if unable to open file
49     except IOError, message:
50        print "Error reading file:", message
51
52     # handle exception parsing file
53     except SAXParseException, message:
54        print "Error parsing file:", message
55
56  if __name__ == "__main__":
57     main()
```

Fig. 16.16 SAX-based parsing example. (Part 2 of 3.)

```
Enter a file to parse: boxes.xml
Enter tag to search for: box

<box> started
   Attributes:
   size = big

   <box> started
      Attributes:
      size = medium
   </box> ended


   <box> started
      Attributes:
      type = small

      <box> started
         Attributes:
         type = tiny
      </box> ended

   </box> ended

</box> ended
```

Fig. 16.16 SAX-based parsing example. (Part 3 of 3.)

Lines 42–43 obtain the name of the XML document to parse and the tag name to locate. Line 46 invokes **`xml.sax`** function ***`parse`***, which creates a SAX parser object. Function **`parse`**'s first argument is either a Python file object or a filename. The second argument passed to **`parse`** must be an instance of class **`xml.sax.ContentHandler`** (or a derived class of **`ContentHandler`**, such as **`TagInfoHandler`**), which is the main callback handler in **`xml.sax`**. Class **`ContentHandler`** contains the methods (Fig. 16.15) for handling SAX events.

If an error occurs during the opening of the specified **`file`**, an **`IOError`** exception is raised, and line 50 displays an error message. If an error occurs while parsing the file (e.g., if the specified XML document is not well-formed), **`parse`** raises a ***`SAXParseException`*** exception, and line 54 displays an error message.

Our example overrides only two event handlers. Methods **`startElement`** and **`endElement`** are called when start tags and end tags are encountered. Method **`startElement`** (lines 16–31) takes two arguments—the element's tag name as a string and the element's attributes. The attributes are passed as an instance of class ***`AttributesImpl`***, defined in **`xml.sax.reader`**. This class provides a dictionary-like interface to the element's attributes.

Line 21 determines whether the element received from the event contains the tag name that the user specified. If so, line 22 prints the start tag, indented by **`depth`** spaces, and line 24 increments **`depth`** by **`3`** to ensure that the next tag printed indented further.

Lines 29–31 print the element's attributes. The **for** loop first obtains the attribute names by invoking the ***getNames*** method of **attributes**. The loop then prints each attribute name and its corresponding value—obtained by passing the current attribute name to the ***getValue*** method of **attributes**.

Method **endElement** (lines 34–39) executes when an end tag is encountered and receives the end tag's name as an argument. If **name** contains the tag name specified by the user, line 38 decreases the indent by decrementing **depth**. Line 39 prints that the specified end tag was found.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 16.17: boxes.xml           -->
4  <!-- XML document used in Fig. 16.16 -->
5
6  <boxlist>
7
8    <box size = "big">
9       This is the big box.
10
11      <box size = "medium">
12         Medium sized box
13         <item>Some stuff</item>
14         <thing>More stuff</thing>
15      </box>
16
17      <parcel />
18      <box type = "small">
19         smaller stuff
20         <box type = "tiny">tiny stuff</box>
21      </box>
22
23   </box>
24
25 </boxlist>
```

Fig. 16.17 XML document used in Fig. 16.16.